Rapport projet TAL
Détection Automatique de la Langue d'un Texte

GALLIENNE Romane
PEREIRA Cindy

Juin 2020

L3 Linguistique Informatique Université de Paris

Table des matières

Élément clé de la majorité des applications informatiques, la reconnaissance de langue, ou Language Identification (LI ou LID), est un domaine de recherche qui a vu le jour il y a une cinquantaine d'années pour des besoins de traduction. Aujourd'hui, avec l'explosion d'internet à usage tant personnel qu'entrepreneurial, sans bien sûr oublier la recherche, le besoin de programmes de détection de langue efficaces est plus que d'actualité. Correction orthographique automatique, requêtes faites aux moteurs de recherche, classement de documents, ou encore traduction, aucune de ces tâches ne saurait être exécutée sans un tel programme.

1 Notions fondamentales

La détection automatique de langue consiste, pour un programme informatique, à reconnaître la langue naturelle utilisée dans un texte ou à l'oral. Il peut donc s'agir, comme nous l'avons dit plus haut, d'une requête dans un moteur de recherche, d'une traduction, etc. pour l'écrit ; ou, dans le cas de l'oral, d'applications de commandes vocales telles que Siri, Alexa, OK Google, etc.

Il est important de noter que dans ce domaine, nous faisons la différence entre langues naturelles et langages.

Ainsi, les langues naturelles (auxquelles appartient la langue des signes) sont des langues apprises naturellement par les humains répondant à des règles communes qu'étudient les disciplines réunies sous le nom de linguistique. Plus concrètement, dans toutes les sociétés, l'enfant apprend à parler progressivement et naturellement au contact de son entourage. Il acquiert de ce fait des bases syntaxiques (i-e de construction de phrases) sans avoir besoin de leçons particulières.

Du fait de l'objet de notre projet, nous nous concentrerons ici spécifiquement sur la reconnaissance de langues naturelles dans les textes. Par conséquent, nous ne traiterons ni de la langue des signes, ni de l'oralité des langues naturelles.

2 Pourquoi la recherche en reconnaissance de langue naturelle est-elle encore nécessaire ?

2.1 La recherche en reconnaissance automatique des langues naturelles

Si l'homme peut, grâce à la connaissance de sa propre langue, reconnaître automatiquement sa langue native, voire distinguer instinctivement l'origine large d'une langue - par

exemple, en voyant un idéogramme, en déduire qu'il s'agit d'une langue asiatique -, un ordinateur en est bien incapable si on ne le lui a pas appris.

Le but des recherches en reconnaissance des langues est justement de parvenir à un programme qui puisse reconnaître toutes les langues naturelles existantes, ce qui va bien au-delà des compétences d'une seule personne, aussi hyperpolyglotte soit-elle.

Malgré les apparences, de nombreux problèmes subsistent encore et parmi ceux-ci, le fait que les recherches liées à ce sujet se soient dispersées entre plusieurs disciplines (Natural Language Processing, Data Mining, Machine Learning, etc.) qui ne communiquent pas leurs résultats [1].

2.2 Les problèmes subsistants

Il est utile de faire un point sur les problèmes subsistant dans la reconnaissance automatique des langues car ce sont des problématiques auxquelles nous allons certainement devoir faire face dans la mise au point de notre propre programme ou, du moins, que nous devons prendre en considération.

Aujourd'hui, les résultats des programmes reconnaissant la langue d'un texte atteignent quasiment la perfection uniquement si l'on considère un nombre restreint de langues naturelles, en utilisant un très grand nombre de textes d'apprentissage et si le texte est intégralement écrit dans la même langue. Cela impose donc de nombreuses contraintes dont il est nécessaire de s'affranchir.

L'un des avantages en reconnaissance automatique des langues par rapport à d'autres disciplines liées, est que nous pouvons nous affranchir de la problématique d'une collection de données spécifique à un domaine car en reconnaissance de langue naturelle, nous ne cherchons pas à reconnaître un type de support (journal, livre, etc.) mais la langue utilisée, ce n'est donc pas tant le vocabulaire qui va nous aiguiller que l'agencement des caractères.

En reconnaissance de langue, nous utilisons un étiquetage spécifique pour chaque langue. Or, lorsqu'un document est rédigé en plusieurs langues (citations, expressions, inclusion de passage, etc.), il n'est aujourd'hui pas possible d'associer deux jeux d'étiquettes, chacun appartenant à une langue différente, pour un même texte. Il faut pour que cela soit possible, pouvoir d'abord segmenter le texte en parties distinctes - chacun contenant une langue différente - pour pouvoir leur appliquer indépendamment le jeu d'étiquettes.

Avec la méthode que nous utilisons pour ce projet, nous ne devrions pas rencontrer de problème d'étiquetage, à moins que la proportion de la langue différente dans le texte ne soit trop élevée.

Nous nous attacherons pour ce projet à élaborer un programme de reconnaissance écrite

1. D'où l'intérêt de l'étude de Jauhiainen T., Lui M., Zampieri, M. Baldwin T., Lindén K (2019) dont le but est de donner un aperçu de l'état des recherches et des problématiques actuelles.

des langues naturelles suivantes : allemand, anglais, espagnol, français et portugais. En fonction de nos avancées, nous envisageons d'ajouter d'autres langues.

3 Méthode

3.1 Objectifs

Notre programme répond à deux objectifs :
— Reconnaître la langue d'un texte déjà écrit dans un fichier.
— Reconnaître la langue d'un texte entré par l'utilisateur dans le terminal.

Il est également capable d'évaluer la performance de notre détecteur de langue. Le code est écrit en Python.

3.2 Corpus d'entraînement

Notre corpus est constitué à partir d'articles du site Wikipedia dont nous avons extrait le contenu en txt à l'aide d'un programme écrit au préalable. En effet, ces derniers sont libres de droit et ont l'avantage considérable d'être traduits dans de nombreuses langues différentes.

La description complète de notre corpus est donnée dans la Partie 4.1 de ce rapport.

3.3 Programme Statistique : La méthode N-Gramme

Initiée en 1974 par Rau qui a obtenu 89% de précision dans la distinction de 53 caractères entre l'anglais et l'espagnol, la méthode N-gramme est ensuite utilisée par Church en 1985 (tri-gramme) et associée au modèle de Bayer pour déterminer la langue de certains mots. C'est finalement la méthode TextCat de Canvar et Trenkle (1997) qui a imposé l'utilisation de la méthode N-gramme pour la détection automatique des langues et cela reste, encore aujourd'hui, la référence dans ce domaine [2].

Pour l'implémentation du programme, nous transformons le texte en un dictionnaire de N-gramme. Nous avons fait le choix de ne pas vectoriser et de calculer les similarités entre des dictionnaires. La raison est expliquée en Partie 5.2 de ce rapport.

La méthode N-gramme permet de modéliser la distribution de caractères dans une langue ou un texte et d'en mesurer la fréquence.

Il s'agit, dans le cas d'une méthode bigramme, d'associer les caractères deux à deux (caractère i + caractère $i-1$), ce qui nous permet de calculer la fréquence à laquelle ces deux

2. Disponible en Open-Source, il réunit des modèles pour 76 langues mais n'est plus actualisé.

caractères se suivent dans une langue.

Ainsi, pour la phrase “le chat et le chien dorment”, si nous segmentons la phrase deux caractères par deux caractères en utilisant la méthode N-gramme, nous obtenons la séquence de chaînes de caractères suivante :

“le”, “e-”, “-c”, “ch”, “ha”, “at”, “t-”, ”-e”, ”et”, ”t-”, ”-l”, ”le”, ”e-”, ”-c”, ”ch”, ”hi”, ”ie”, ”en”, ”n-”, “-d”, “do”, “or”, “rm”, ”me”, ”en”, ”nt”.

Notons qu’appliquer la similarité vectorielle a deux contraintes fortes :

— Il est nécessaire d’avoir un texte brut dans le cas d’importation de fichier.
— Plus le vecteur est grand, plus le coût est important.
— Les langues très proches (ex : Portugais et Espagnol, Portugais et Français) sont moins bien distinguées.

3.4 Programme de reconnaissance de la langue d’un texte : Similarité vectorielle

Pour chaque corpus, nous ajoutons les N-grammes dans un dictionnaire, puis nous leur associons les fréquences correspondantes : si le N-gramme est déjà présent dans le dictionnaire, nous ajoutons 1 à sa fréquence ; si le N-gramme n’est pas présent, nous ajoutons une clé au dictionnaire avec une fréquence de 1.

Ainsi, pour reprendre l’exemple de la phrase “le chat et le chien dorment”, nous obtenons le dictionnaire suivant :

{”le” : 2, ”e-” : 2, ”-c” : 2, ”ch” : 2, ”ha” : 1, ”at” : 1, ”t-” : 2, ”-e” : 1, ”et” : 1, ”-l” : 1, ”hi” : 1, ”ie” : 1, ”en” : 2, ”n-” : 1, ”-d” : 1, ”do” : 1, ”or” : 1, ”rm” : 1, ”me” : 1, ”nt” : 1}.

Une fois le dictionnaire de la phrase complété, nous le comparons par similarité vectorielle aux autres dictionnaires (complétés de la même manière) du corpus de langues.

On peut calculer la similarité entre deux textes par deux méthodes : en calculant la distance euclidienne, ou en calculant la similarité par cosinus.

On calcule la distance euclidienne entre les deux dictionnaires en calculant la norme de la différence des deux dictionnaires, soit la formule pour deux dictionnaires x et y :

$$Dist(x, y) = ||x - y||$$

Ainsi, plus la distance entre les deux dictionnaires est faible, plus la similarité est importante.

La similarité par cosinus est le rapport entre le produit scalaire des deux dictionnaires et leurs normes respectives, soit la formule pour deux dictionnaires x et y :

$$cos\ \theta = \frac{xy}{||x||*||y||}$$

Le cosinus calculé doit être compris entre -1 et 1. Plus le résultat se rapproche de -1, moins les textes sont similaires et plus il se rapproche de 1, plus ils sont similaires. Si le cosinus est proche de 0, cela signifie que les deux dictionnaires sont orthogonaux : les textes n'ont aucun N-gramme en commun.

Nous discuterons plus loin de la différence de résultats entre ces deux méthodes.

L'évaluation du programme est discutée dans la Partie 4.2 du rapport.

4 Expérience et résultats

4.1 Corpus d'entraînement

Afin d'obtenir un nombre conséquent de textes pour chaque langue, nous avons décidé de récupérer les pages correspondant à la description de chaque pays du monde. Cependant, nous obtenions ainsi des corpus de tailles différentes et donc non utilisables tels quels. Nous avons donc ajouté des pages aux sujets divers et variés pour avoir des corpus contenant (à peu près) le même nombre de caractères pour chaque langue.

	Nombre de fichiers	Nombre de lignes	Nombre de mots	Nombre de caractères
Allemand	399	301 168	955 006	11 138 385
Anglais	195	318 845	1 010 987	11 133 853
Espagnol	113	186 247	1 167 787	11 137 551
Français	597	302 304	984 707	11 134 745
Portugais	241	187 648	1 173 101	11 132 506

Table 1: Contenu de notre corpus de langues

4.2 Évaluation

Tout programme mis au point doit faire l'objet d'une évaluation afin d'en connaître la qualité. L'approche la plus commune d'évaluation est d'avoir le même nombre de documents pour chaque langue (dont on connait la langue) et de les soumettre au programme puis de regarder la proportion de documents correctement étiquetés par le programme.

Nous avons donc créé un corpus de test pour toutes les langues testées : allemand, anglais, espagnol, français et portugais. Chacune de ces langues possède neuf fichiers dont la distribution est la suivante : trois fichiers de 100 caractères, trois fichiers de 200 caractères et trois fichiers de 500 caractères.

Nous calculons ensuite la précision ou valeur prédictive positive grâce à la formule suivante :

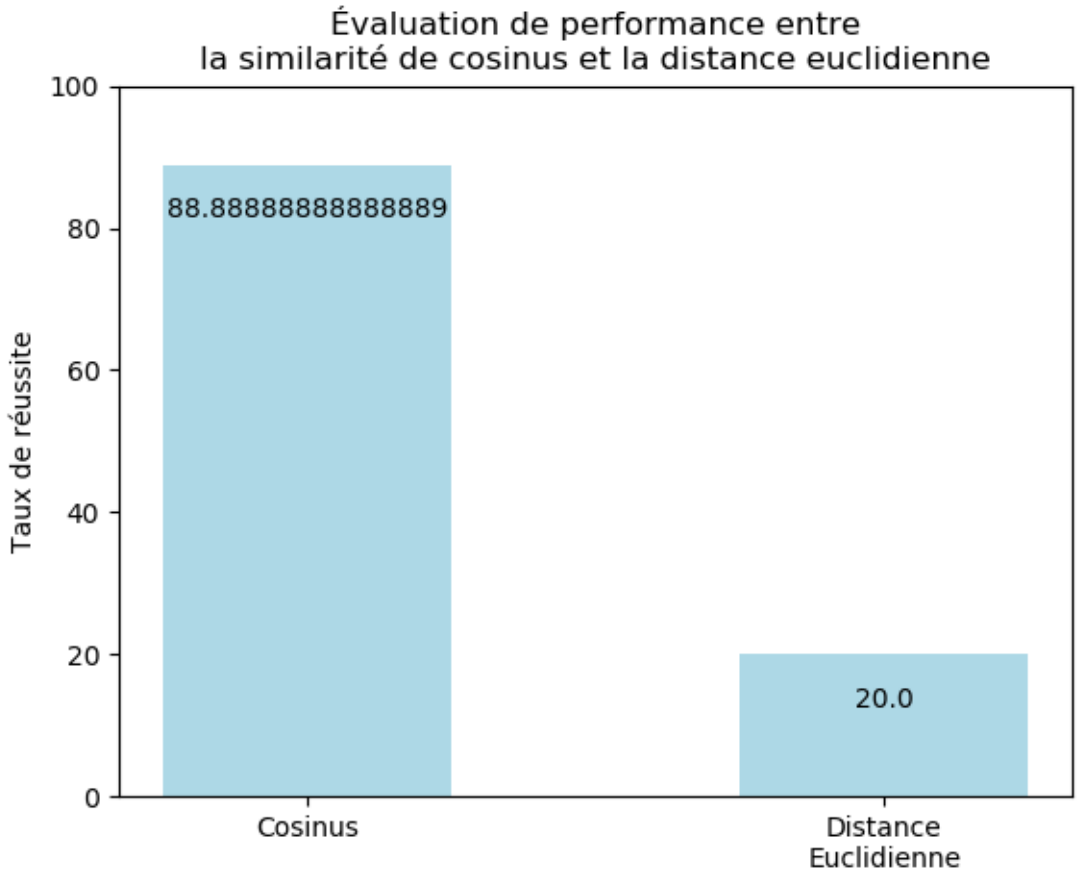

FIGURE 1: Évaluation de performance entre les deux méthodes en utilisant des bigrammes

$$Précision = \frac{Nombre\ de\ documents\ correctement\ étiquetés}{Nombre\ de\ documents}$$

Pour faciliter l'évaluation de notre programme, nous avons décidé de classer les textes selon un indice dans une liste : les indices 0 à 9 correspondent aux textes allemands, les indices de 9 à 17 correspondent aux textes anglais, 18 à 26 aux textes espagnols, 27 à 35 les textes français et 36 à 44 les textes portugais.

Si le résultat du progamme pour le texte à l'indice i de notre liste de textes correspond à la langue présente à l'indice i de la liste de langues, alors le résultat est correct. Nous ajoutons alors une valeur "True" dans une liste de résultats. Cette liste permettra de savoir exactement quels textes ont été correctement étiquetés. Cela nous sera utile afin d'évaluer la performance en fonction de la taille du texte.

Nous avons testé les deux méthodes présentées dans le début de ce rapport (distance euclidienne et similarité cosinus) afin de comparer leurs performances et ainsi choisir celle qui serait la plus efficace pour la suite de nos tests. Ces évaluations ont toutes deux été faites en utilisant des bigrammes (Figure 1).

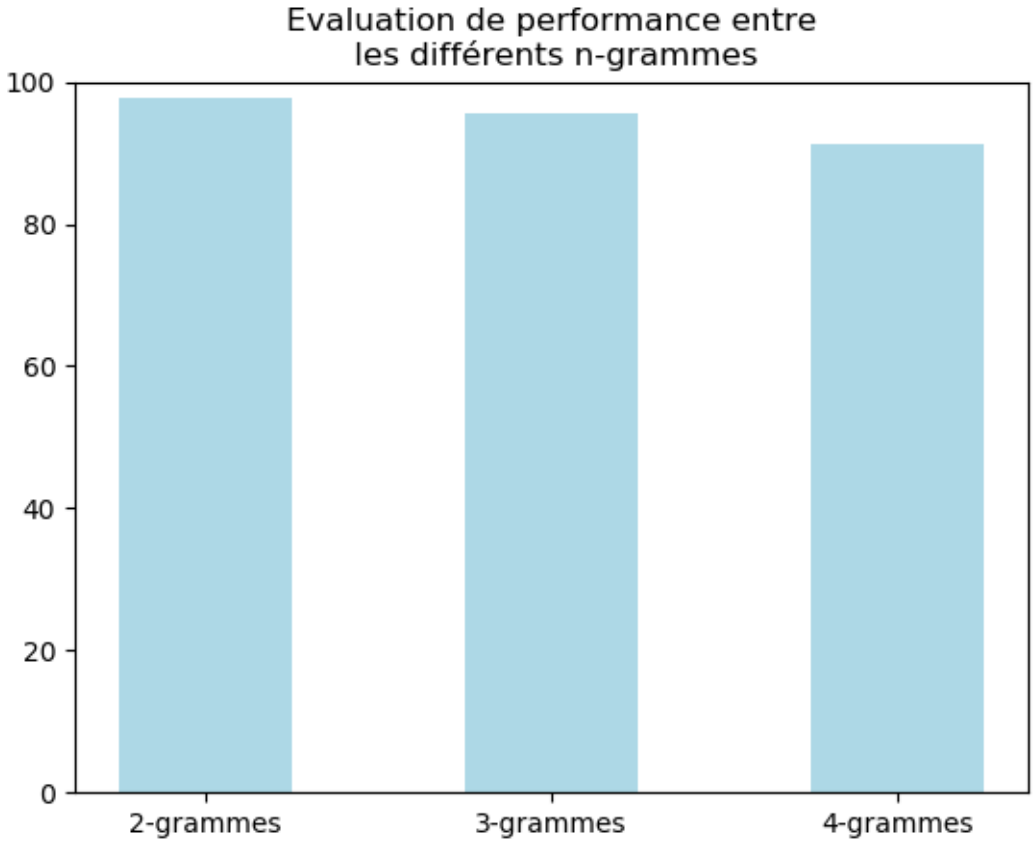

FIGURE 2: Évaluation de perfomances entre les différents N-grammes

Nous observons qu'il existe une grosse différence entre ces deux méthodes ; la similarité cosinus semble bien plus efficace que la distance euclidienne. Par conséquent, nous avons fait le choix de n'utiliser que la similarité cosinus pour la suite des évaluations.

Nous avons ensuite voulu évaluer la différence de performance dans l'utilisation de différents N-grammes. Nous avons alors comparé les résultats de bigrammes, trigrammes et 4-grammes (Figure 2).

Nous remarquons que les bigrammes semblent plus performants que les deux autres N-grammes. Cependant, la différence n'étant pas forcément significative, nous avons voulu comprendre comment évoluait la performance de chacun de ces N-grammes en fonction de la taille du texte.

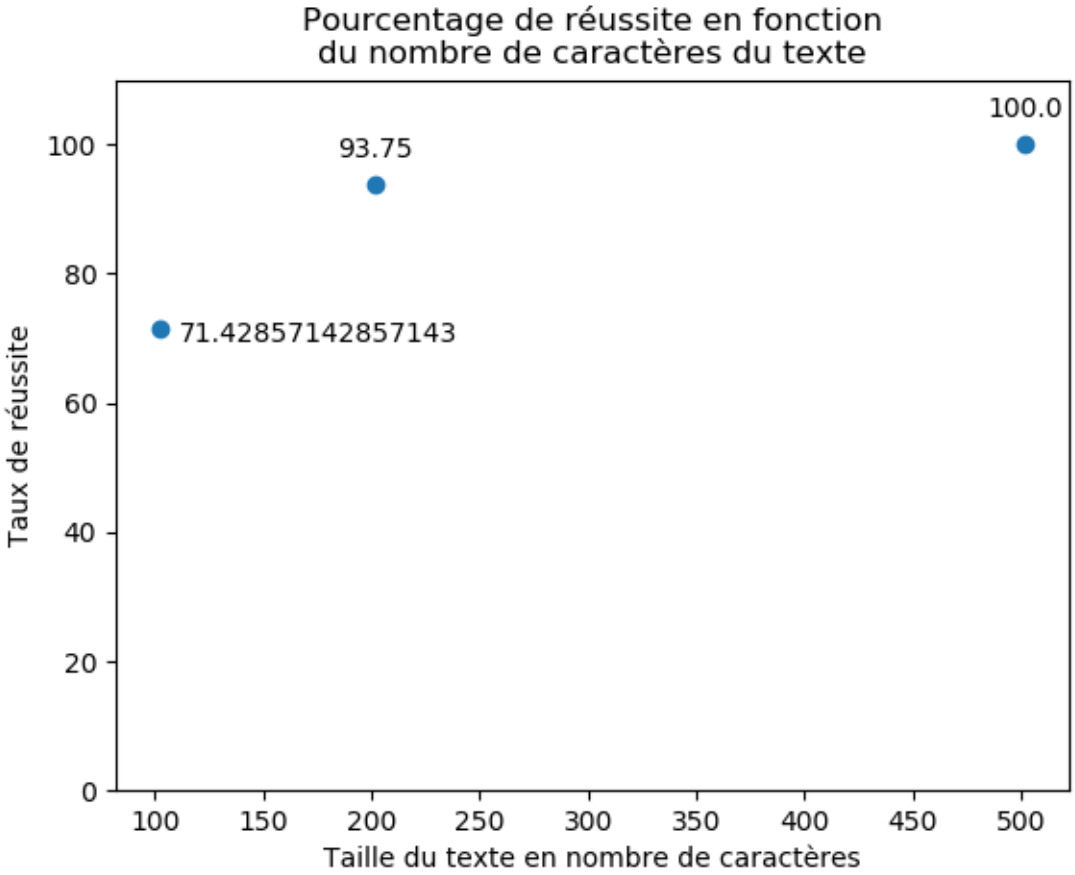

FIGURE 3: Pourcentage de réussite de notre programme utilisant des bigrammes en fonction de la taille en caractères du texte

On remarque que de manière générale, le programme étiquette correctement les textes en utilisant des bigrammes (Figure 3). Notamment, les grands textes (500 caractères) sont étiquetés correctement dans 100% des cas. Plus le texte est petit cependant, moins la précision est élevée. Par exemple, les textes de 200 caractères obtiennent une précision de 93.75% contre 71.43% pour les petits textes (100 caractères).

Lorsqu'on utilise des trigrammes (Figure 4), la précision est assez élevée quelle que soit la taille du texte. On remarque tout de même qu'avec des trigrammes, le programme est moins performant sur les textes moyens et longs (respectivement 81.25% et 93.33%) qu'avec des bigrammes, cependant, il est plus performant sur les textes courts (85.71%).

L'utilisation de 4-grammes (Figure 5) ne semble pas adaptée au vu des résultats plutôt bas obtenus (50.0% pour les petits textes, 56.25% pour les textes moyens). Cependant, on remarque tout de même que la précision a tendance à augmenter lorsque les textes sont de taille plus conséquente (86.65% pour les textes de 500 caractères).

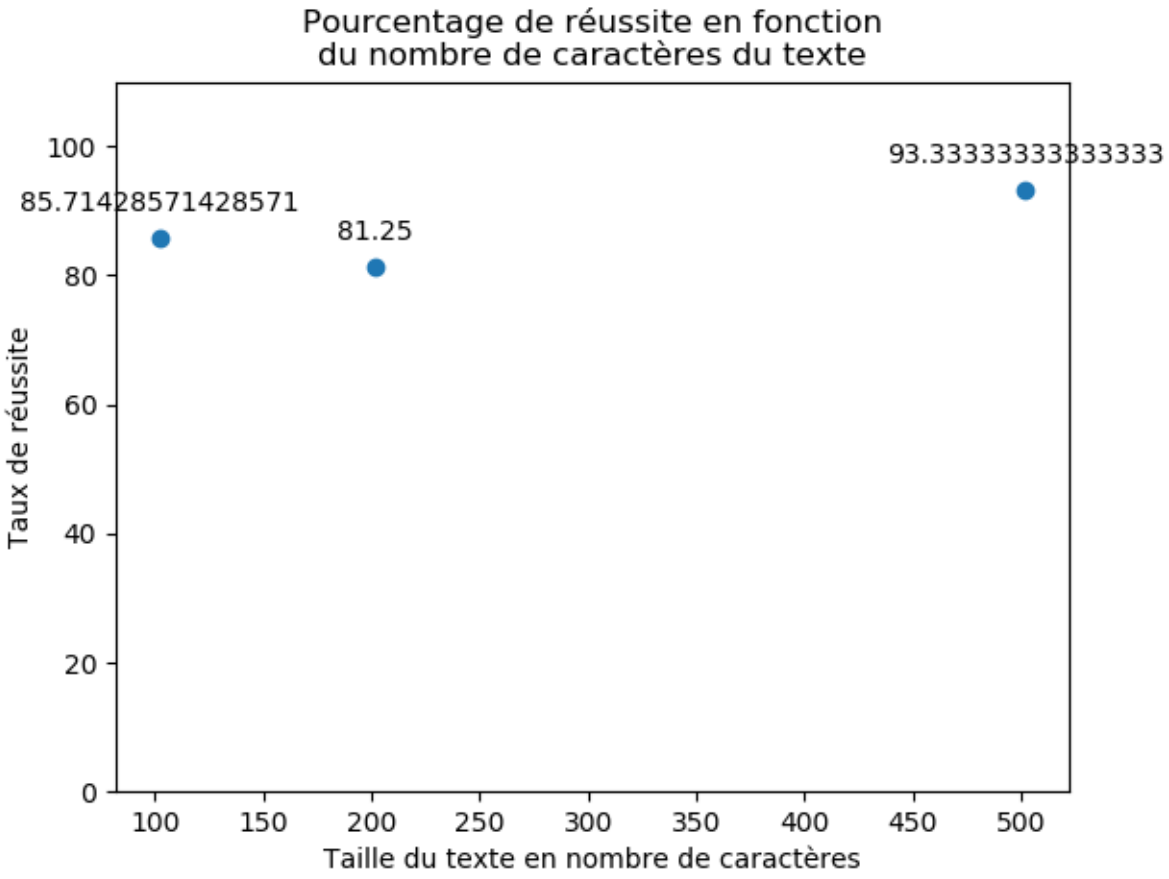

FIGURE 4: Pourcentage de réussite de notre programme utilisant des trigrammes en fonction de la taille en caractères du texte

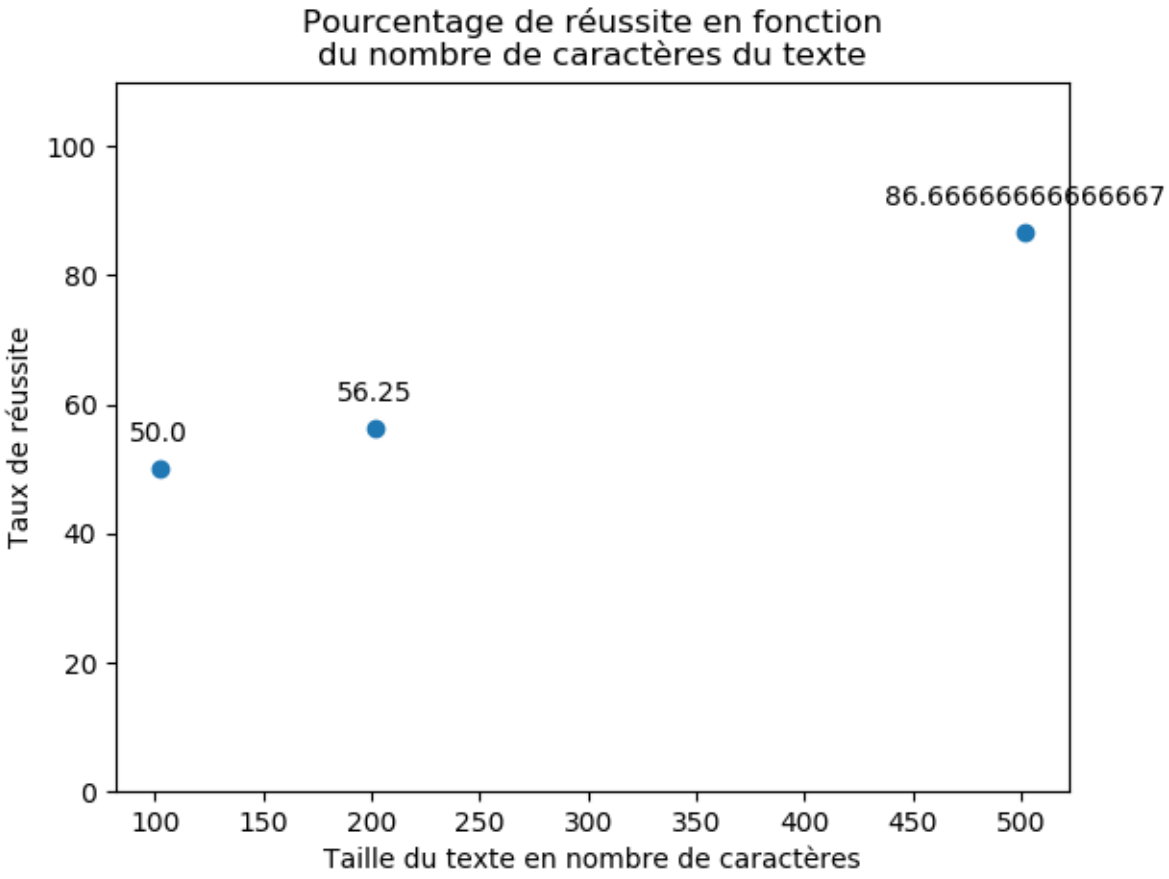

FIGURE 5: Pourcentage de réussite de notre programme utilisant des 4-grammes en fonction de la taille en caractères du texte

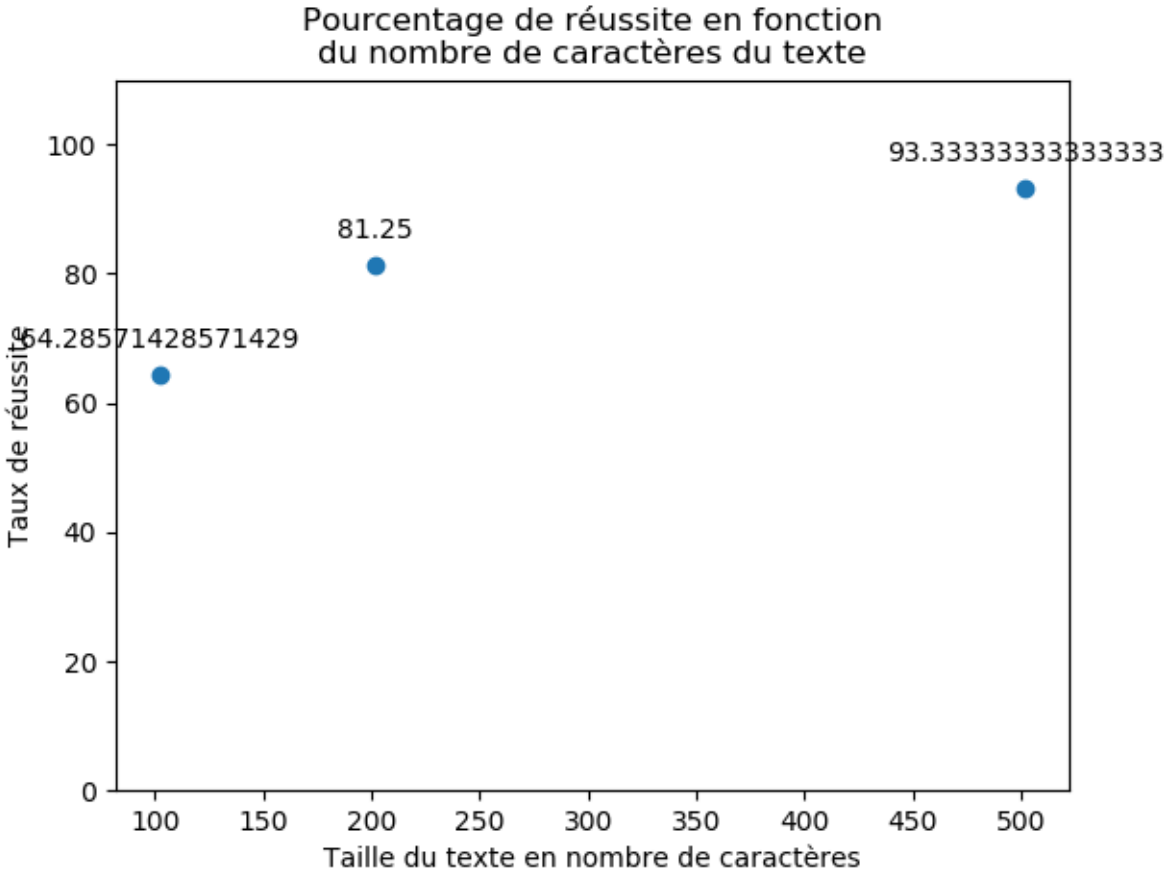

FIGURE 6: Pourcentage de réussite de notre programme utilisant des unigrammes en fonction de la taille en caractères du texte

Nous avons décidé de ne pas essayer sur des n-grammes supérieurs à 4, car les performances nous semblaient diminuer lorsque n devenait plus grand.

Par curiosité, nous avons voulu voir la performance de notre programme lorsque l'on génère des unigrammes (Figure 6). On remarque que les performances sont moins élevées qu'avec des bigrammes, mais que les résultats sont tout de même plutôt corrects.

Nous avons également testé notre programme sur des phrases entrées directement dans le terminal. Nous remarquons que notre programme peut manquer de précision sur de très courtes phrases d'un ou deux mots de type "Bonjour" ou encore "Je m'appelle". Ce manque de précision dépend des langues ; en effet le Portugais par exemple est très mal reconnu avec des bigrammes mais l'est plus facilement avec des trigrammes. On a pu constater qu'une langue est mieux reconnue lorsque des combinaisons de caractères spécifiques sont présentes une ou plusieurs fois dans la phrase entrée.

5 Partie informatique

5.1 Composition du programme

Notre programme se compose de cinq fichiers : *Evaluation*, *Fonctions*, *Graphes*, *Detection* et *VectorisationCorpus*.

Dans le fichier *Fonctions*, il y a les différentes fonctions permettant de récupérer le texte d'un fichier ou de l'ensemble des fichiers d'une langue, nettoyer le texte, créer le dictionnaire de fréquence de N-grammes et calculer les deux méthodes de similarités.

Le fichier *Detection* est le fichier principal du programme. Il permet de calculer les similarités (cosinus et distance euclidienne) entre un texte et chaque langue. Il récupère ensuite la meilleure similarité et renvoie la langue détectée. Il peut être appelé depuis le src avec la commande suivante :

```
python3 Detection.py n texte
```

Où n est la taille des n-grammes et *texte* peut être soit un nom de fichier, soit le texte à étiqueter entré directement.

Le fichier *Evaluation* permet d'évaluer la précision de notre programme. Il peut être appelé depuis le src avec la commande suivante :

```
python3 Evaluation.py n
```

Où n est la taille des n-grammes.

Le fichier *Graphes* permet de générer des graphiques pour évaluer les différents paramètres en appelant le fichier *Evaluation*. Il peut être appelé depuis le src avec la commande suivante :

```
python3 Graphes.py n
```

Où n est la taille des n-grammes.

Le fichier *VectorisationCorpus* a permis de vectoriser les textes pour le corpus d'entraînement et de créer et sérialiser les dictionnaires. Nous avons déjà sérialisé des dictionnaires de unigrammes, bigrammes, trigrammes et 4-grammes pour chaque langue du corpus. Il n'est donc normalement pas nécessaire de le lancer. Cependant, si besoin, il peut être lancé depuis le src avec la commande :

```
python3 VectorisationCorpus.py n
```

Où n est la taille des n-grammes.

5.2 Difficultés rencontrées et choix d'implémentation

Lorsque nous avons commencé à travailler sur le projet, nous utilisions des dictionnaires dont chaque valeur de clé était un vecteur unitaire. Pour chaque texte ou langue, nous calculions ce dictionnaire en fonction des n-grammes rencontrés. Ensuite, à partir de celui-ci, nous calculions le vecteur du texte en sommant les vecteurs de chaque n-gramme rencontré.

Cette méthode impliquait de nombreux problèmes. En effet, les dictionnaires récupérés associaient à chaque n-gramme un vecteur de la taille du nombre de n-grammes différents rencontrés dans le corpus, ce qui les rendait très volumineux. Par conséquent, la sérialisation était compliquée car il fallait compresser les données, et même ainsi, les fichiers étaient très lourds. La désérialisation à chaque lancement du programme prenait du temps et le traitement de si grosses données ralentissait considérablement notre programme.

Malgré l'absence d'une évaluation précise, cette méthode nous semblait cependant renvoyer de bons résultats, mais les problèmes engendrés étaient trop importants pour que nous puissions garder cette approche. Nous avons donc adopté la méthode exposée en Partie 3 de ce rapport.

Dans le fichier *Graphes*, nous importons l'ensemble du fichier *Evaluation*. Cela est expliqué par le fait que *Graphes* appelle *Evaluation* et crée les graphiques à partir des données calculées par ce dernier. Par conséquent, nous avions besoin de l'entièreté du fichier *Evaluation*, et non uniquement de ses fonctions.

Dans les fichiers *Fonctions* et *VectorisationCorpus*, il existe deux fonctions de récuperation de textes assez répétitives. Cependant, les types renvoyés sont différents et ces fonctions ne peuvent pas être utilisées dans le même contexte. Dans le fichier *Fonctions*, on récupère les textes du dossier *CorpusTest* pour une langue donnée et on les stocke dans une liste de String. Ce type nous permettra, dans l'évaluation, de garder les textes séparés et de savoir quel texte a été correctement identifié.

Dans *VectorisationCorpus*, on récupère les textes du dossier *Corpus* pour une langue donnée et on les stocke dans un String. Ce type est suffisant car on ne souhaite pas traiter les textes un à un mais comme un unique texte. En effet, dans ce fichier, l'important est que l'ensemble des textes soit de taille équivalente pour chaque langue.

Il est possible de choisir la taille souhaitée pour les n-grammes au lancement du programme. Cependant, nous avons choisi de ne garder que les dictionnaires correspondant aux unigrammes, bigrammes, trigrammes et 4-grammes. En effet, comme expliqué dans la Partie 4.2, l'utilisation de n-grammes plus grands ne nous a pas semblée pertinente. Il est toutefois possible de créer d'autres dictionnaires en lançant le fichier *VectorisationCorpus*.

6 Manuel utilisateur

Pour Detection, il est absolument nécessaire d'entrer un n. Pour les autres fonctionnalités du programme, si n n'est pas précisé, 2 est pris par défaut. Se

placer dans le répertoire src avant de lancer le programme.

```
cd L3GroupeBDetection/src
```

6.1 Vectorisation des dictionnaires : facultatif

Lancer le programme avec le fichier VectorisationCorpus.py
Argument facultatif : n (taille des n-grammes)
Exemple d'utilisation :

```
python3 VectorisationCorpus.py 5
```

6.2 Détection de la langue d'un texte

Lancer le programme avec le fichier Detection.py

Arguments nécessaires :
— n (taille des n-grammes)
— texte (nom d'un fichier au format txt ou texte directement entré)

Exemples d'utilisations :

```
python3 Detection.py 2 Bonjour
python3 Detection.py 2 train.txt
```

6.3 Évaluation et graphiques

1. Évaluation : facultatif
   Lancer le programme avec le fichier Evaluation.py
   Argument facultatif : n (taille des n-grammes).
   Exemple d'utilisation :

   ```
   python3 Evaluation.py 2
   ```

2. Graphiques
   Peut être lancé sans avoir lancé Evaluation au préalable.
   Lancer le programme avec le fichier Graphes.py
   Argument facultatif : n (taille des n-grammes).
   Exemple d'utilisation :

   ```
   python3 Graphes.py 2
   ```

7 Bibliographie

— Jauhiainen T., Lui M., Zampieri M., Baldwin T., Lindén K. (2019). **Automatic Language Identification in Texts : A Survey**. *Journal of Artificial Intelligence Research*. 65, 675-782

— Claveau V. (2019). **Vectorisation, Okapi et calcul de similarité pour le TAL : pour oublier enfin le TF-IDF**. *Journal of Artificial Intelligence Research*. TALN - Traitement Automatique des Langues Naturelles, Jun 2012, Grenoble, France. Hal-00760158

— Wikipedia. Précision et rappel [en ligne]. (page consultée le 30/03/2020). `https://fr.wikipedia.org/wiki/Pr%C3%A9cision_et_rappel`

— Wikipedia. Modèle vectoriel [en ligne]. (page consultée le 30/03/2020). `https://fr.wikipedia.org/wiki/Mod%C3%A8le_vectoriel`